PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 1/00

A1

(11) International Publication Number: WO 98/50843

(43) International Publication Date: 12 November 1998 (12.11.98)

(21) International Application Number: PCT/US98/08927

(22) International Filing Date: 1 May 1998 (01.05.98)

(30) Priority Data:
08/850,601 2 May 1997 (02.05.97) US

(71) Applicant: NETWORK ASSOCIATES, INC. [US/US]; 3965 Freedom Circle, Santa Clara, CA 95054 (US).

(72) Inventor: SPILO, Michael, L.; 248 East 31st Street, New York, NY 10016 (US).

(74) Agent: FELLER, Mitchell, S.; Darby & Darby P.C., 805 Third Avenue, New York, NY 10022–7513 (US).

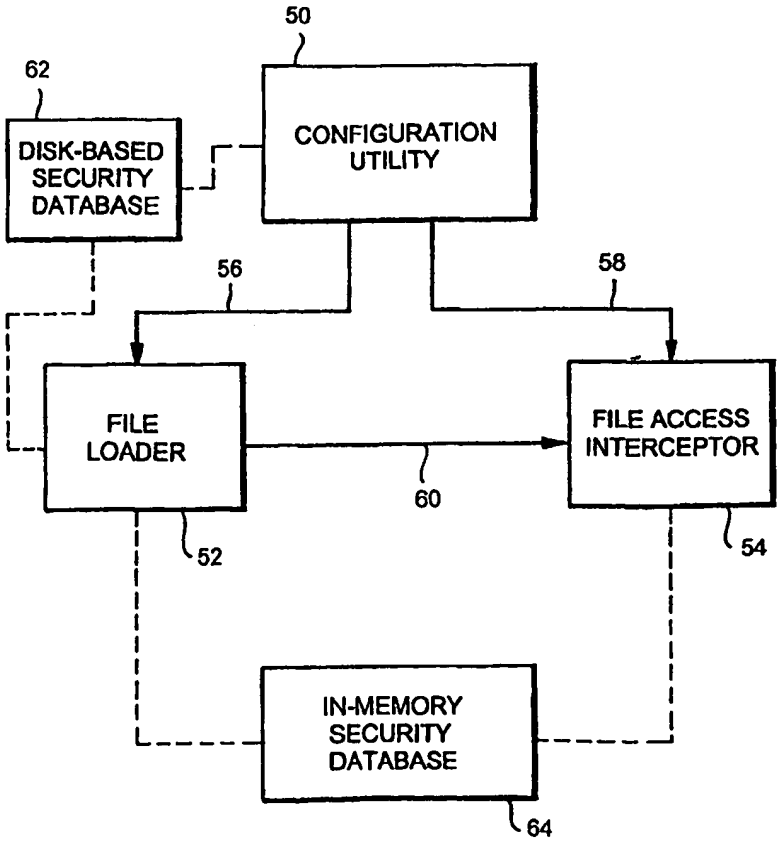
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: PROCESS–LEVEL DATA SECURITY SYSTEM

(57) Abstract

A process–level data security system for use on a computer includes three primary components: a configuration utility, a file loader, and a file access interceptor. The configuration utility is used to establish a security context for each desired program file. A program having a security context associated therewith, after having been loaded into the computer's memory by the file loader of the invention, can access only limited information on a mass storage device coupled to the computer. The file access interceptor of the invention handles all file requests issued by the program, and permits or rejects such requests according to the security context.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Paso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

PROCESS-LEVEL DATA SECURITY SYSTEM

The invention relates to a system and method for securing stored computer data against unauthorized access, and more particularly to a computer operating system enhancement whereby certain processes and tasks can be prevented from accessing specified areas of a computer's mass storage system.

BACKGROUND OF THE INVENTION

The need for data security has long been recognized in computer systems, especially those that are used by multiple individuals and those that are connected by network to the outside world.

Traditionally, security provisions are implemented in computer operating systems to guard against access by unauthorized individuals. For example, when a computer is used by multiple individuals, each user's confidential data can be protected, if desired, from access by others.

This "user-level" security scheme typically operates as follows. Each user is provided with a user ID (or "login name") and a password. Users must "log in" to the computer using the login name and password before any access or usage is allowed. After a successful login is made, access continues to be limited in a number of ways.

Every file stored on the computer is associated with an "owner." The owner of a file can change his own and other users' privileges and permissions with respect to the

file. For example, a user can indicate that a specific program file he owns can be read, written to, and executed by him, but only read by others. Similarly, that same user might be forbidden from reading, writing, or executing certain files owned by another user sharing the system.

In such a system, a system administrator or "super-user" typically has access to all files regardless of owner and regardless of security settings. The system administrator should be trusted not to breach the security provisions otherwise established on the system.

A typical computer security implementation of the type described above is used in the UNIX operating system. See generally, Robert Felps, Illustrated UNIX System V/BSD (1992), pp. 24-26 (re the UNIX file system in general) and 79-84 (re the UNIX chmod command).

This scheme is most useful when a number of people are using and storing files on a single computer system, and when a network connection is used only for remote logins.

Today's most common variety of computer is a personal computer. The need for user-level security is less pronounced, as there is often a one-to-one relationship between computers and users. However, different security concerns have arisen.

Personal computers are often connected to networks, for example the "Internet." This network connection is often accessed through a "client" program, which is pre-programmed to send and receive certain kinds of information over the network in response to user commands and requests. For example, the Netscape Navigator and Microsoft Internet Explorer client programs, which are known generally as "browsers," are often used to view information on the "World Wide Web," a collection of special-purpose servers connected to the Internet. While these browsers are programmed to retrieve data in response to user requests, a large amount of information can be passed back and forth between the user's computer (by way of the client program) and Web

servers, based on commands received from the Web servers. This is a potential security issue.

Moreover, a new generation of Web browsers allows a form of computer program to be sent from the Web server to a user's computer for execution. These programs can take a variety of forms, the two most common of which presently are "applets" written in the Java language (originated by Sun Microsystems) and so-called "ActiveX controls" implemented (by way of a specification provided by Microsoft) to run under the Microsoft Windows operating system. These Java applets and ActiveX controls can be caused to run on the user's computer system with little or no user intervention. Although the Java language, in particular, is usually implemented with a certain level of data security in mind, it is possible for such security to be circumvented and undesired data to be transmitted over the network.

Even in the absence of multiple users or a network connection, it is recognized that certain programs are capable of causing accidental or malicious damage to other files stored on the same computer. Accordingly, to ensure the safe operation of such programs, it is desirable to provide an isolated environment in which they can run. This has long been recognized in the context of isolating running programs from each other in memory. See Matt Pietrek, Windows Internals (1993), pp. 207-208; and Andrew Schulman, Unauthorized Windows 95 (1994), pp. 298-301.

However, an additional benefit can be realized by isolating programs from each other in terms of long-term storage. If a program cannot access other programs' areas on a disk, then those areas cannot be overwritten, either accidentally or deliberately. Ill-behaving programs can be limited to a small area of disk space to prevent damage to other areas.

In summary, while a person using a computer might want access to everything on the computer, it would be useful to be able to restrict the access of any specific executing program to only certain limited information.

Accordingly, there is a need to limit storage access for certain programs. Rather than a user-level security system, which permits or prohibits access based on whether an individual user has the proper permission, such a security system would be a process-level system. A process-level security system would permit or prohibit access to disk-based information based on the identity of the program or module requesting the access.

SUMMARY OF THE INVENTION

The process-level data security system of the invention addresses the security issues recognized in a single-user networked environment.

Every running task can have associated therewith a security context. Pursuant to the security context, the task is allowed access to only a limited area of the computer's mass storage. A configuration utility is used to establish the security context for each task, as well as a default security context for all other tasks.

A task can also inherit a security context from a "parent" task. In this way, a program generally having more liberal security privileges cannot be used as an instrumentality by other programs to allow access to otherwise forbidden data. This provision prevents ill-behaved Java applets, for example, from breaching security.

Consequently, the security limitations imposed by the invention can prevent unauthorized access to, and subsequent transmission of, confidential data over a network connection. It can also prevent damage caused by an application attempting to write data to locations it should not have access to, either accidentally or maliciously.

In contrast to traditional security schemes, the invention allows security parameters to be established on a per-process rather than per-user, basis.

To implement this security scheme, the invention has three primary components. A configuration utility component is used to manually establish security contexts for any desired application programs; such security contexts are stored in a disk-based security database. A file loader component is used to correlate information in the disk-based security database to the processes currently being run by the computer's CPU. A file

access interceptor component uses information kept in memory by the file loader to determine whether individual attempts at file access should be permitted or refused.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other objects of the invention will become apparent from the detailed description below and the accompanying drawings in which:

FIGURE 1 is a block diagram of a computer system capable of employing the process-level data security system of the invention;

FIGURE 2 is a block diagram illustrating the relationships within a storage hierarchy as seen by the invention;

FIGURE 3 is a block diagram illustrating the relationships among the three components of the invention;

FIGURE 4 is a flowchart describing the operation of the configuration utility of the invention;

FIGURE 5 is a flowchart illustrating the operation of the file loader of the invention; and

FIGURE 6 is a flowchart illustrating the operation of the file access interceptor of the invention.

DETAILED DESCRIPTION OF THE INVENTION

The invention is described below, with reference to detailed illustrative embodiments. It will be apparent that a system according to the invention may be embodied in a wide variety of forms. Consequently, the specific structural and functional details disclosed herein are representative and do not limit the scope of the invention.

A computer system according to the invention is typically a standalone workstation or personal computer, with components as illustrated in FIGURE 1. A central processing unit (CPU) 10 is coupled to a mass storage system 12, such as a hard

disk drive. The CPU 10 is also coupled to a network 14. The network 14 is represented in FIGURE 1 as a single functional block; however, it should be recognized that the network 14 can comprise a large number of additional computer systems coupled for communication with the CPU 10. The CPU 10 is also coupled to an input/output (I/O) system 16, through which a user is capable of interacting with or instructing the CPU 10.

The CPU 10, in most computers, runs an operating system program (not shown). The operating system program manages the interaction among the mass storage system 12, the network 14, and the I/O system 16. Any application software running on the CPU 10 makes requests of the foregoing subsystems through the operating system program. For example, if an application program needs access to data stored on the mass storage system 12, a request is generally made to the operating system program, which will then control the mass storage system 12 so as to provide the requested data to the CPU 10 for use by the application program. Similarly, an application program can request, based on an instruction from the user via the I/O system 16, data to be transferred from the network 14; this request, too, will be handled by the operating system program.

On so-called "PC compatible" computers, such as those using the Intel 80x86 series of microprocessors, the most common operating system program presently in use comprises a combination of Microsoft Disk Operating System ("MS-DOS") and Microsoft Windows, although MS-DOS can be used without Windows. Several versions of Windows are in use, such as Windows 3.1 and Windows 3.11 (together called "Windows 3.1x") and Windows 95. For purposes of this invention, it should be noted that, although a separate version of MS-DOS is not necessary to run Windows 95, it is in fact bundled with its own integrated version of MS-DOS.

As discussed above, one purpose of the operating system program is to handle transactions between the CPU 10 and the mass storage system 12. Such transactions are typically in the form of file operations. For example, an application can request that a particular file present on the storage system 12 be opened for reading, and that certain

data be read. Based on that request, the operating system program will identify the desired file, open it for reading, and transfer the appropriate data to the CPU. The operating system maintains a database of files present on the storage system 12, with information on where such files are specifically stored. An application program need not concern itself with such details; it can merely specify a data file by name to the operating system. The operating system will then locate the desired file based on the specified name.

On PC-compatible computers, file access can be made by way of Interrupt 21h ("INT 21h") services. This mechanism has long been used in MS-DOS programs. In real mode (a CPU mode in which MS-DOS operates), invoking a "software interrupt," in this case Interrupt 21h, causes program execution to transfer through an Interrupt Vector Table (IVT) to an interrupt handling routine (part of the operating system program). In this manner, an application program gets the attention of the operating system program, so that a file access request can be made. Typically, the operating system's handling routine for Interrupt 21h reads certain processor registers to determine the nature of the operation desired by the application program that included the INT 21h instruction, performs the operation, and then returns control to the application program.

When Windows is utilized with MS-DOS, this method is altered somewhat. Invocation of an Interrupt 21h causes the Windows Virtual Machine Manager (VMM), a component of Microsoft Windows, to be notified by way of a transfer through an Interrupt Descriptor Table. The VMM will handle the interrupt in one of at least two ways: the VMM can perform the requested service itself (or make additional requests to various components of Windows) and return to the calling program; or the VMM can "reflect" the request to MS-DOS, which in Windows 3.1x and Windows 95 is present in memory concurrently with Windows, and have MS-DOS ultimately perform the operation and return control to the calling application program. See generally Schulman, Unauthorized Windows 95, chapter 8. The method used is dependent on several factors,

including the version of Windows in use, whether 32-bit file access is enabled in certain versions, and the nature of the particular file operation requested.

It should be noted that Interrupt 21h calls can originate in DOS programs running under Windows, 16-bit Windows programs (intended for use with Windows 3.0 or 3.1x), and Virtual Device Drivers (VxDs) called by 32-bit Windows programs, such as the VWIN32 VxD contained in VMM32.VXD, a component of Windows 95.

Examples of various Interrupt 21h operations are: function 3Ch (create file), function 3Dh (open file), function 3Fh (read file), function 39h (create directory), function 40h (write file), function 41h (delete file), function 47h (get current directory), function 4Bh (execute program), and function 6Ch (extended open/create). Under Windows 95, a large number of additional functions are available to facilitate, for example, the use of "long filenames" which previous versions of MS-DOS and Windows did not support. As discussed above, the invention intercepts attempts to access these functions, and many others, to ensure that the proper security parameters are followed before Windows or MS-DOS is allowed to perform the desired file operation.

File access can also be made by several 16-bit Windows functions contained in the KERNEL module. So-called "16-bit functions" are callable from programs intended for use under Windows 3.1x, as well as from programs written for Windows 95. An application program running under Windows can transfer control directly to these functions without issuing an Interrupt 21h request. These functions are present in Windows 3.1x and Windows 95. For example, the OpenFile function is passed a filename to request that a disk file be opened. The GetTempFileName function returns an original unique filename for use as temporary storage. The _lread, _lwrite, _llseek, and _lclose functions perform various operations, as suggested by the function names, on already opened files.

Moreover, as suggested above, use of standard file input/output functions present in most language compilers will cause a combination of API functions and Interrupt 21h requests to be issued. Accordingly, all of the API functions and Interrupt 21h functions

must be intercepted by the present invention. This should be done in a way other than by trapping Interrupt 21h, as there is a way to execute MS-DOS functions without invoking a software interrupt (e.g. by using the Windows API functions DOS3Call and NoHookDOSCall).

In Windows 95, an additional method of file access is available. An applications program interface ("API") comprising a large number of file access functions (among other types of functions) is made available for use by 32-bit (i.e. Windows 95 specific) application programs. These functions also provide access to data stored on the mass storage system 12, but do not utilize the Interrupt 21h scheme discussed above. Rather, an application program transfers control to Windows directly, which then invokes the proper file access routine from the appropriate VxD. Upon completion of the operation, Windows transfers control back to the application program.

Examples of the Windows 95 file access API include the functions CreateFile, CreateDirectory, DeleteFile, ReadFile, WriteFile, GetTempFileName. All of these functions, which are contained in the Windows 95 KERNEL32 module, perform the operations suggested by their names. Requests made of all of these Windows 95 API functions, as well as many others, must be intercepted by the invention to ensure proper enforcement of security parameters.

FIGURE 2 illustrates an exemplary hierarchy present on a mass storage system such as that of storage system 12. In order to make access to files stored on a mass storage system more efficient, these files can be organized into groups known as "directories." In this way, data files having similar content, usage, or characteristics can be grouped together for a user's convenience. Directories are actually data files that contain information specifying where data files are stored on the block storage device. They can often be hierarchical; in other words, directory files can point to other directory files, which in turn point to data files. The location of a file within a directory hierarchy can be specified by means of a "pathname" to the file, which indicates the name of each directory traversed as well as the filename.

FIGURE 2 shows a root directory 20. The root directory can point to files (such as a first file 22 and a second file 24) and additional directories (such as a first directory 26, a second directory 28, and a third directory 30). In turn, the directories, such as the first directory 26, can contain additional files and directories (such as a first subdirectory 32 and a second subdirectory 34). The third exemplary directory 30, as shown in FIGURE 2, includes a third subdirectory 36, which in turn includes a fourth subdirectory 38 and a fifth subdirectory 40. Consequently, any file or directory on the mass storage system 12 can be uniquely identified by specifying the directories traversed beginning at the root directory. For example, a data file 42 contained in the fourth subdirectory 38 can be identified by specifying the name of the root directory 20, the name of the third directory 30, the name of the third subdirectory 36, the name of the fourth subdirectory 38, and the name of the data file 42.

In practical usage, these directory and file specifications are concatenated into the "pathname" discussed above. For example, if the root directory is named "D:" (which is in the standard form for a root directory name under MS-DOS and Windows, as the letter "D" identifies a particular storage unit), the third directory is named "Program Files," the third subdirectory is named "Netscape," the fourth subdirectory is named "Navigator," and the data file is named "readme.txt," then the full pathname will be "D:\Program Files\Netscape\Navigator\readme.txt." As is standard in MS-DOS and Windows usage, the names of the directories are separated by the backslash character "\".

Referring now to FIGURE 3, the invention as shown has three primary functional components: a configuration utility 50, a file loader 52, and a file access interceptor 54. Data generated by the configuration utility is used by the file loader as shown by a first arrow 56; data generated by the configuration utility is also used by the file access interceptor as shown by a second arrow 58. Information from the file loader is also used by the file access interceptor as shown by a third arrow 60. Two databases are also used by the invention. A disk-based security database 62, stored on the mass storage system

12 (FIGURE 1), is created by the configuration utility 50; it comprises a list including security information for each of various processes capable of running on the computer system. The file loader 52 uses the disk-based security database 62 to create an in-memory security database 64, quickly accessible to the CPU 10 (FIGURE 1), for the processes currently running. The file access interceptor 54 uses the information in the in-memory security database 64 to permit or prohibit file access operations according to the invention.

The operation of the configuration utility 50 (FIGURE 3) is described in detail in connection with FIGURE 4. The configuration utility is an interactive software application capable of running on the computer system to be protected. For each application the user desires to prevent from accessing certain data, the user first selects the filename of the desired program file (step 70).

The user then selects the files and directories he wants the program file to be permitted access to (step 72). The user can then select the files and directories he explicitly wants the program file to be denied access to (step 74). The security context for the program file can include both criteria: certain files and directories can be explicitly included, while others are explicitly excluded. For both inclusion and exclusion, separate "read," "write," and "execute" permissions can be set. The user can then select a desired action for all other (unspecified) storage areas (step 76). Specifically, the user can permit access, deny access, or prompt for action with regard to other directories and files, including those directories and files not in existence at the time the configuration utility 50 is run. Upon completion of these steps, the desired security parameters for the program file are then stored in memory as a security context record (step 78).

If the user wishes to create security contexts for further programs (step 80), then the above steps are repeated. Otherwise, the user can create a default security context for all unspecified program files. It should be noted that not every program file needs to have an explicit security context; the default context will be used for all other programs.

The default context is created in the same manner as above: the user selects areas to include (step 82), areas to exclude (step 84), and an action to take for unspecified areas (step 86). The default context is then stored in memory (step 88).

All of the security context records in memory, including the default context, are then stored in the disk-based security database 62 (step 90). The records will later be loaded into memory as necessary by the file loader component of the invention. Finally, the in-memory security database 64 is updated, if necessary, to reflect the changes made to the disk-based security database 62 (step 92). If the user has not modified the security context for any process currently running, then the step of updating the in-memory security database is not necessary.

The user can re-run the configuration utility as desired to update process security contexts. For example, if the user later learns that a particular network-related program is improperly accessing a confidential area of his disk, then that area can be added to the excluded directories of the program's security context.

The invention is operative after the user selects an initial set of desired security contexts with the configuration utility. Thereafter, whenever a process is initiated, the appropriate security context data will be brought into memory by the file loader and utilized by the file access interceptor to permit or decline file access attempts.

The operation of the file loader is shown in FIGURE 5. The file loader intercepts attempts to load a new process, and updates the in-memory security database accordingly.

The file loader intercepts the Windows KERNEL function WinExec and Interrupt 21h function 4Bh (EXEC) (step 100). As discussed above, Interrupt 21h function 4Bh is used to load and execute a new file. Under Windows, this function is not "reflected" to MS-DOS; it is handled by a special routine within the Windows KERNEL.

Both the WinExec function and Interrupt 21h function 4Bh invoke the Windows KERNEL function LoadModule. See Schulman, Unauthorized Windows 95, p. 490; see also Pietrek, Windows Internals, pp. 229-272. The file loader of the invention allows

this to proceed (step 102), but immediately also establishes a security context for the new process. One operation performed by the LoadModule function is to establish a task database ("TDB") for the new process. Every separate process has its own TDB, even if it shares code with other processes. The structure of an exemplary TDB under Windows 3.1 is described in Pietrek, Windows Internals, at pp. 226-228.

After the LoadModule function completes, but before a new task has an opportunity to request any file operations, the invention determines whether a new TDB was created (step 104). A TDB is created only if the module loaded represents a new task. If no TDB was created, the file loader component returns control to the operating system (step 106) for normal operation.

Otherwise, the file loader reads the contents of the new TDB (step 108). The name of the module stored in the TDB is checked against the disk-based security database (step 110). If there is a match (step 112), and a security context has been created for the task, then the corresponding security context record is retrieved from the disk-based security database (step 114) and stored in the in-memory security database (step 116).

If there was no match (step 112), and no security context was created for the task, then the default security context record is retrieved from the disk-based security database (step 118) and stored in the in-memory security database (step 120). As noted above, a security context does not need to be established for every single task capable of running on the computer. Moreover, new untested applications (e.g. those just installed) can be given a narrow range of disk access in this way.

The file loader component then determines what task called for the new process (step 122). This information is found in the "parent task" record in the new task's TDB. The security context of the parent task (taken from the in-memory security database) is then combined with the new task's security context read from the disk-based security database (step 124). Specifically, any excluded disk areas are combined (e.g. via a "union" set operation); included disk areas are used only to the extent they overlap (e.g.

via an "intersection" set operation). For unspecified areas, the more restrictive context prevails (i.e. "prevent access" prevails over "prompt for action," and "prompt for action" prevails over "allow access").

The combined security contexts provide for a type of "inheritance." A newly created task can not have a higher level of file access than the program that created it. For example, if Netscape Navigator loads a helper application, it generally should not have a higher privilege than Navigator does. However, if the helper application is used in standalone mode, with no network connection, then it makes sense for it to have a broader range of permissions.

Space is allocated within the in-memory security database (step 126). The combined security context is then written to the new space in the in-memory security database (step 128). The file loader then returns control to the operating system (step 130) for normal operation.

A further function of the file loader is to determine when a task has completed. When that occurs, the file loader removes the task's information from the in-memory security database. If the task is restarted later, the appropriate record in the in-memory security database will be recreated by the method described above.

After the invention has been installed into memory, and the appropriate operating system file operations have been intercepted, any attempt to access a file will be trapped by the mechanism described above (for Interrupt 21h, 16-bit API, and 32-bit API function calls) before the file operation is to be performed. The operation of the file access interceptor is shown in FIGURE 6.

Upon intercepting an eligible call, the interceptor first determines if the requested function implicates file security considerations (step 140). As discussed above, certain Interrupt 21h functions call for disk access to be made; however, others do not. If the intercepted function does not involve file access, the interceptor will return control to the operating system (step 142), allowing the requested operation to continue.

In particular, the file access interceptor component of the invention must intercept and handle requests to open files. As will be discussed in detail below, once a file has been validly opened, and a "handle" identifying the open file is allocated, very little further intervention is required.

The identity of the task making the request is ascertained by the invention (step 144) by means well-known in the art. The in-memory security database is then queried for the permissions applicable to the requesting task (step 146). As discussed above, each running task has a corresponding record in the in-memory security database; this is ensured by the file loader component of the invention.

The nature of the requested operation is then compared against the permissions stored in the in-memory security database for the file in question (step 148). If access is allowable (step 150), then control is passed to the operating system for normal operation (step 152). If not, then a failure condition is indicated (step 154).

In one embodiment of the invention, the failure condition is expressed by returning a code to the application that requested the file operation, indicating that the requested operation was unsuccessful. The file operation will not be performed. The requesting application program will see the failure as an error (e.g. which can be simulated by way of a "disk full" or a "file not found" error), and will handle the error in its usual manner, which may vary from application to application.

In an alternative embodiment, the failure condition causes the file access interceptor to prompt the user for an action. Upon prompting, the user may then confirm the invention's decision to halt the operation, thereby sending an error code back to the requesting application, or may indicate that the invention override the security context on a case-by-case basis. In the latter case, the requested file operation will be performed; optionally, the in-memory security database is updated to reflect the user's decision to allow access to that file.

In an alternative embodiment of the invention, files for which a process does not have access permission are rendered invisible to the application. In this embodiment, several additions to the file access interceptor become necessary.

For example, an application program can infer the existence of certain directories, even if it does not have access to them. For example, if the Netscape Navigator browser program is running from the directory D:\Program Files\Netscape\Navigator, the program can infer the existence of at least the D:\Program Files\Netscape, D:\Program Files, and D:\ (root) directories. This is true even if all other directories are invisible. Some or all of these directories may be forbidden in the Navigator's security context.

There are at least two ways to handle this situation. Preferably, access to invisible but inferred directories can be prohibited, as discussed above. Alternatively, directory names can be changed globally. For example, in the latter scheme, the "D:\Program Files\Netscape\Navigator" directory can be made to look like an "E:\" directory to the Navigator application. This method involves intercepting the Windows functions dealing with pathnames and filenames, and substituting filenames as necessary according to the program's security context. This method has the disadvantage that the same directory or file can appear to have pathnames to separate tasks. Moreover, this scheme might also cause user confusion, as pathnames would be altered by security context and translated later. In other words, when files are specified by way of user entry, an MS-DOS environment variable, the Windows 95 system registry, or an INI file, for example, the filename or pathname must be converted to reflect the security context in which it will be used. This might not always be possible. Consequently, the former approach, in which pathnames are not converted, and access to inferred directory names is prohibited, is preferred.

Although the invention is discussed herein as having three primary components, namely the configuration utility, the file loader, and the file access interceptor, it should be recognized that the components are substantially interdependent. For example, the

file loader is subject to the protection scheme provided by the file access interceptor. If a presently running program attempts to run a new task, the security context for the program might indicate that permission should be declined for the desired task.

Moreover, it should be noted that operating system services other than file operations can be intercepted and handled by various embodiments of the invention. For example, the security context for a given application can include permissions relating to hardware device access. When a request is made to access a particular hardware device, that request can be handled by an interceptor according to the invention and caused to fail, if necessary. As a generalized extension of that principle, application programs can be prevented from loading virtual device drivers (VxDs) or requesting services from VxDs; security contexts can then be established to handle VxD access. These accesses can be intercepted and handled by the interceptor of the invention through means well known in the art.

It will be appreciated that embodiments of the invention may be employed in many different applications to protect against unauthorized access to valuable or confidential data on a mass storage system.

What is claimed is:

1. A method for protecting computer data files from access by unauthorized processes, comprising the steps of:

intercepting an attempt by a process to access a computer data file having a pathname;

ascertaining the identity of the process;

querying a database to retrieve security information corresponding to the process;

comparing the pathname to the security information;

permitting access to the data file based on results of the comparison.

2. The method of claim 1, wherein the database comprises security information corresponding to at least one process.

3. The method of claim 2, wherein the database further comprises default security information.

4. The method of claim 1, wherein the comparing step comprises the substeps of:

determining whether the data file is in a first list of files within the database for which access should be permitted;

determining whether the data file is in a second list of files within the database for which access should be denied;

if the data file is in neither list, determining what action is desired.

5. The method of claim 4, wherein the first list and the second list each correspond to a specified process.

6. The method of claim 4, wherein the first list and the second list each correspond to default security information.

7. The method of claim 1, wherein the security information comprises information corresponding to the process in combination with information corresponding to a parent process.

8. The method of claim 7, wherein the parent process is a task which requested that the process be initiated.

9. The method of claim 1, wherein the permitting step comprises the substeps of:

determining whether access should be permitted;

if access should be permitted, returning control to an operating system program;

if access should not be permitted, causing a failure condition.

10. The method of claim 9, wherein the causing step comprises passing an error code to the process.

11. The method of claim 9, wherein the causing step comprises the substeps of:

prompting the user for an action;

performing the action specified by the user.

12. A system for protecting computer data from unauthorized access on a per-process basis, comprising:

a CPU;

a mass storage system coupled to the CPU for storing the computer data;

an operating system program for controlling interaction between the CPU and the mass storage system;

a disk-based security database stored on the mass storage system comprising a list of permissions; and

a file access interceptor program capable of intercepting attempts by a process to access files on the mass storage system, adapted to permit or deny access based on permissions stored in the disk-based security database.

13. The system of claim 12, further comprising a configuration utility program for establishing at least one security context for an application program.

14. The system of claim 13, wherein the security context comprises a list of files and directories for which access is to be granted.

15. The system of claim 14, wherein the security context further comprises a list of files and directories for which access is to be denied.

16. The system of claim 13, further comprising a file loader program capable of associating the permissions with a process being loaded.

17. The system of claim 16, wherein the permissions associated with the process being loaded comprise a security context for the process.

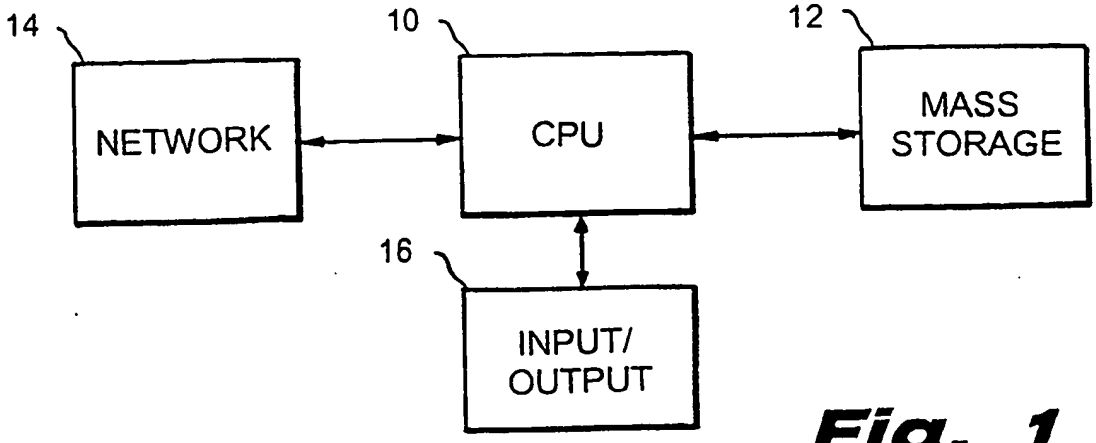

14
NETWORK
10
CPU
12
MASS STORAGE
16
INPUT/ OUTPUT

Fig. 1

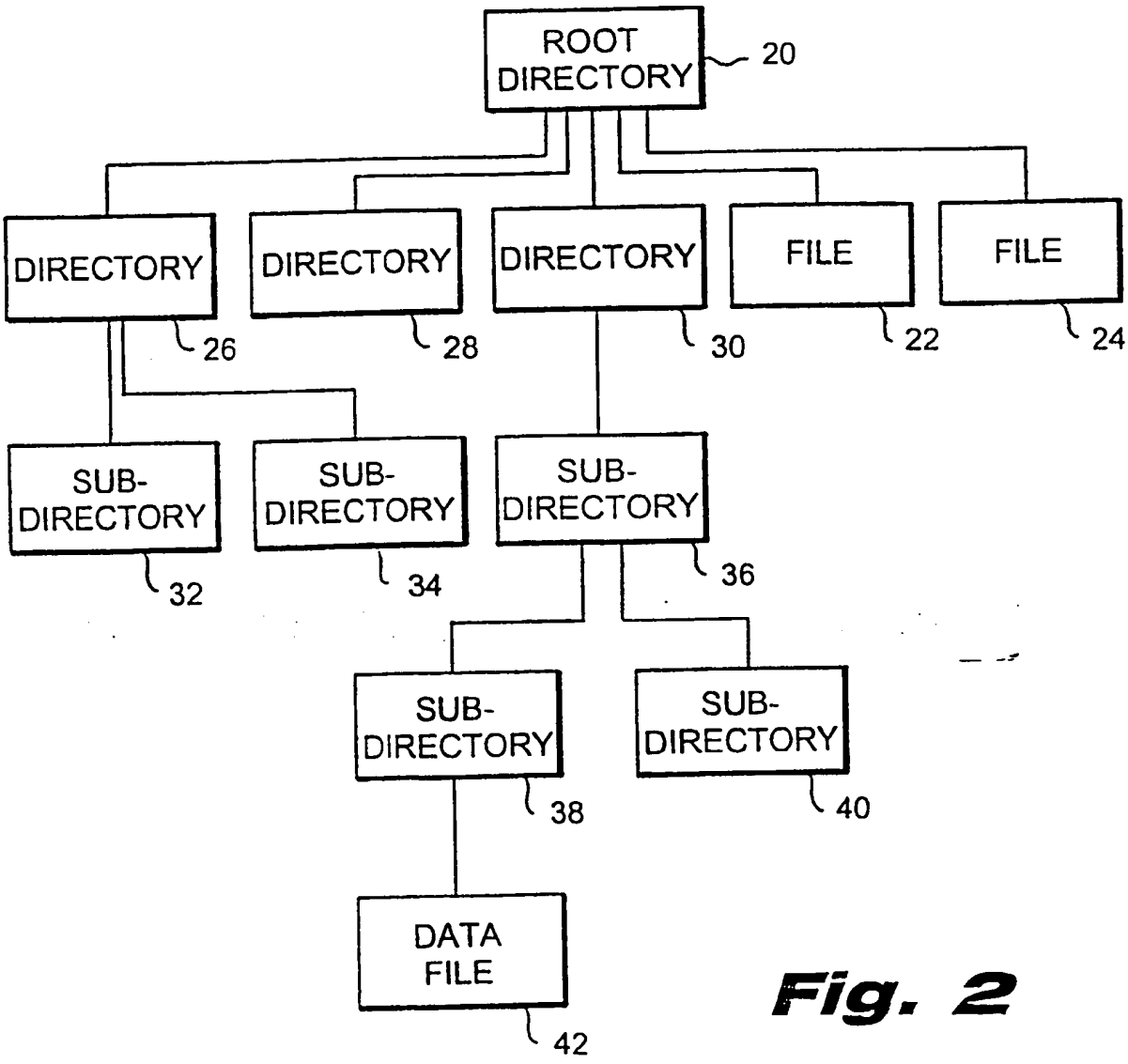

ROOT DIRECTORY
20
DIRECTORY
26
DIRECTORY
28
DIRECTORY
30
FILE
22
FILE
24
SUB-DIRECTORY
32
SUB-DIRECTORY
34
SUB-DIRECTORY
36
SUB-DIRECTORY
38
SUB-DIRECTORY
40
DATA FILE
42

Fig. 2

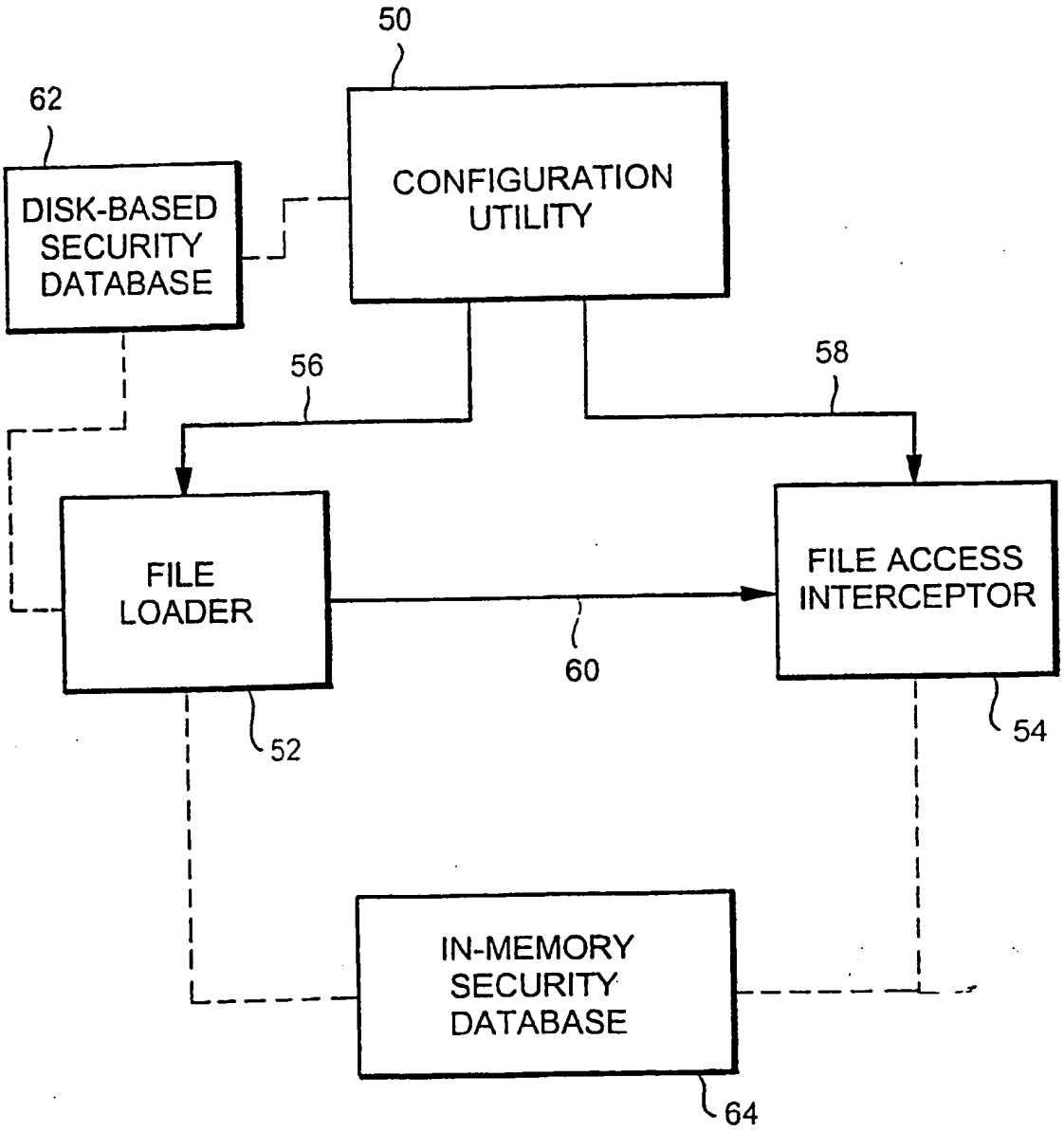
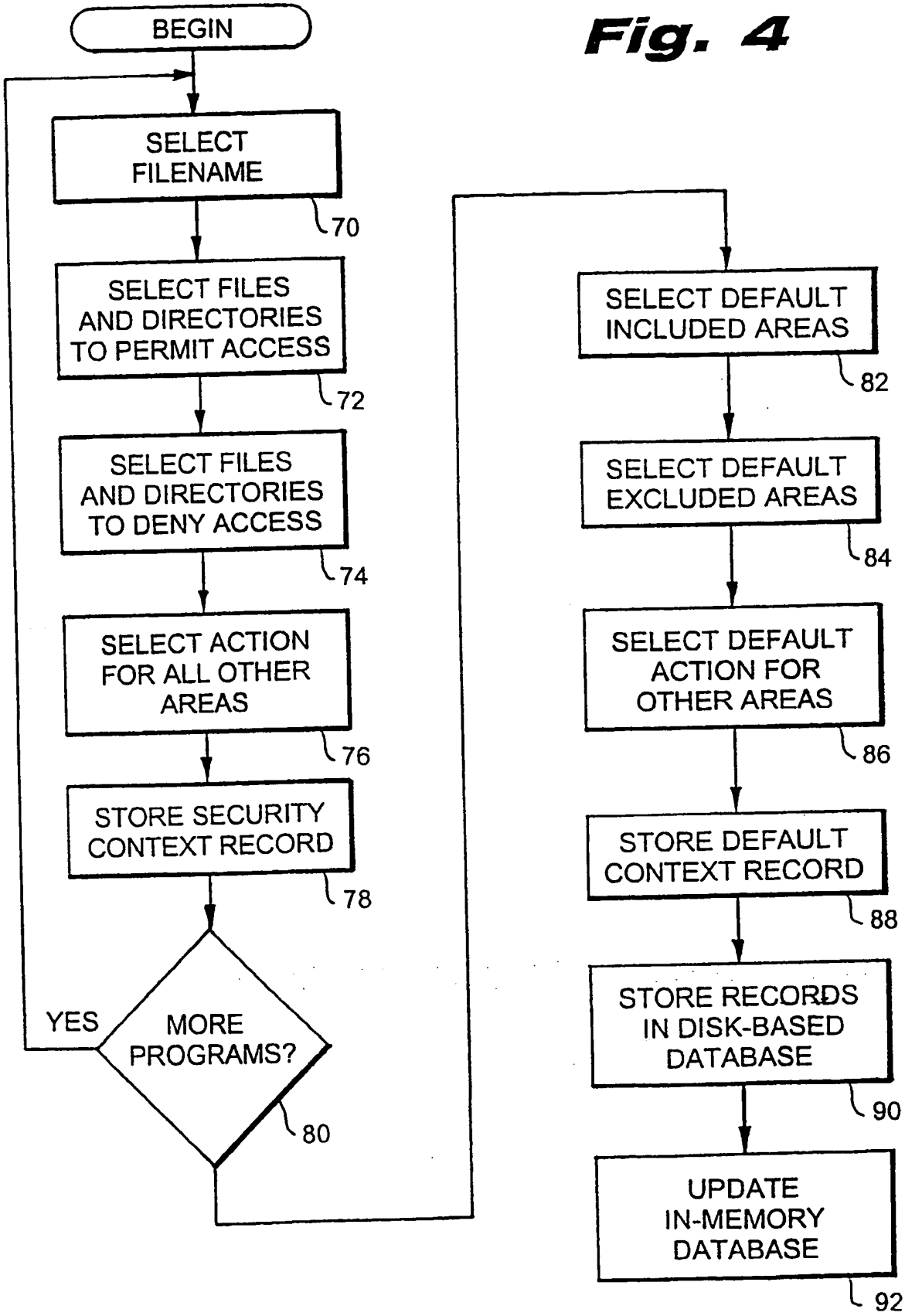

50
62
CONFIGURATION UTILITY
DISK-BASED SECURITY DATABASE
56
58
FILE LOADER
FILE ACCESS INTERCEPTOR
60
52
54
IN-MEMORY SECURITY DATABASE
64

Fig. 3

Fig. 4


BEGIN
SELECT FILENAME
70
SELECT FILES AND DIRECTORIES TO PERMIT ACCESS
72
SELECT FILES AND DIRECTORIES TO DENY ACCESS
74
SELECT ACTION FOR ALL OTHER AREAS
76
STORE SECURITY CONTEXT RECORD
78
YES
MORE PROGRAMS?
80
SELECT DEFAULT INCLUDED AREAS
82
SELECT DEFAULT EXCLUDED AREAS
84
SELECT DEFAULT ACTION FOR OTHER AREAS
86
STORE DEFAULT CONTEXT RECORD
88
STORE RECORDS IN DISK-BASED DATABASE
90
UPDATE IN-MEMORY DATABASE
92

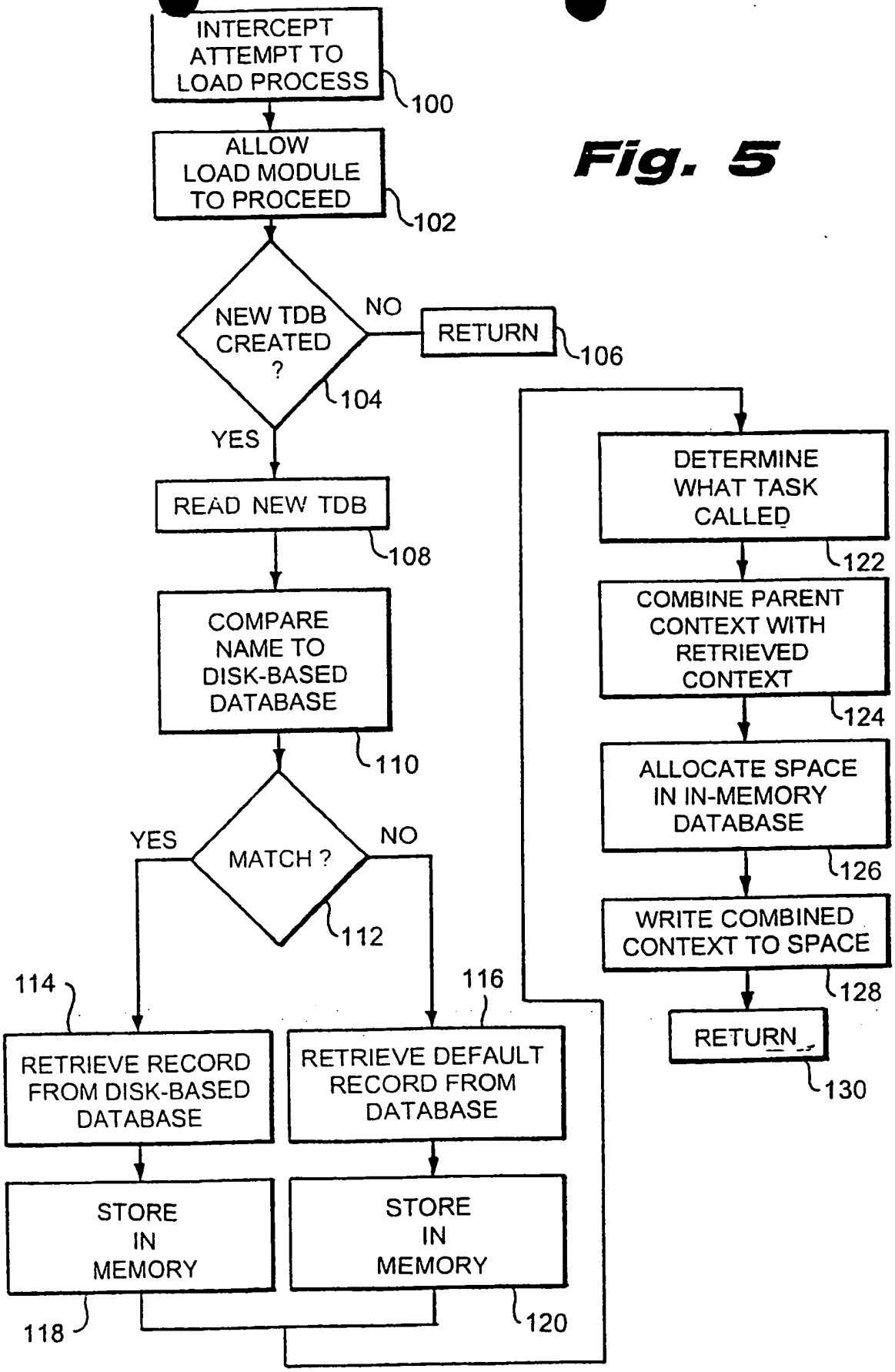

INTERCEPT ATTEMPT TO LOAD PROCESS
100
ALLOW LOAD MODULE TO PROCEED
102
NEW TDB CREATED ?
NO
RETURN
106
104
YES
READ NEW TDB
108
COMPARE NAME TO DISK-BASED DATABASE
110
YES
MATCH ?
NO
112
114
116
RETRIEVE RECORD FROM DISK-BASED DATABASE
RETRIEVE DEFAULT RECORD FROM DATABASE
STORE IN MEMORY
STORE IN MEMORY
118
120
DETERMINE WHAT TASK CALLED
122
COMBINE PARENT CONTEXT WITH RETRIEVED CONTEXT
124
ALLOCATE SPACE IN IN-MEMORY DATABASE
126
WRITE COMBINED CONTEXT TO SPACE
128
RETURN
130

Fig. 5

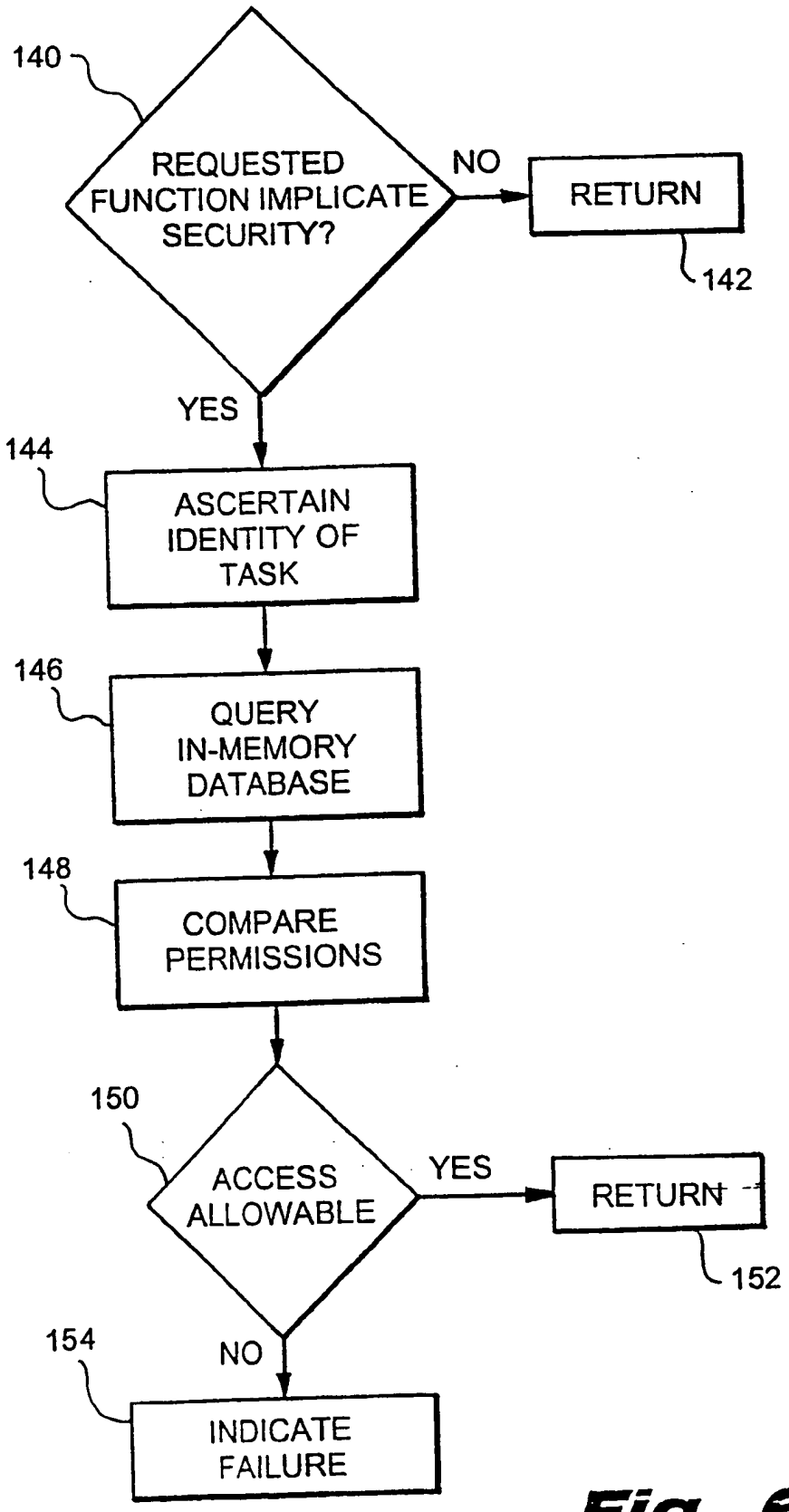

140
REQUESTED FUNCTION IMPLICATE SECURITY?
NO
RETURN
142
YES
144
ASCERTAIN IDENTITY OF TASK
146
QUERY IN-MEMORY DATABASE
148
COMPARE PERMISSIONS
150
ACCESS ALLOWABLE
YES
RETURN
152
154
NO
INDICATE FAILURE

Fig. 6

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 98/08927

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 542 045 A (LEVINE ANATOLI) 30 July 1996 see figure 2 see column 2, line 10 - line 52	1,9,12
A	US 5 163 147 A (ORITA YUKIO) 10 November 1992 see figures 1-5 see column 2, line 53 - column 5, line 55	1,2,4,5, 9-14,16

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 August 1998	20/08/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Weiss, P

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 98/08927

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5542045 A	30-07-1996	NONE	
US 5163147 A	10-11-1992	JP 3088052 A	12-04-1991